1 General Introduction

In the past decades, computing infrastructures (clusters, supercomputers, grids, and clouds) have been continuously evolving and made important contributions to all scientific domains, such as physics, astronomy, medicine, biology etc. Initially, large-scale infrastructures were used to simulate complex phenomena (e.g. weather forecasting, chemical processes). Such simulations are typically compute-intensive (i.e. the time spent applying operations on the data is orders of magnitude higher than reading or writing the data itself). However, more recently, a paradigm-shift has emerged, as many scientific domains have started to produce large amounts of data that need to be analyzed. Common examples of domains that generate large amounts of data are astronomy [2, 99] and bioinformatics [83, 73], which collect telescope observations or DNA sequences.

As opposed to compute-intensive simulations, running applications that iterate through such large amounts of data puts pressure not only on the computing infrastructure's CPUs, but also on its storage devices (i.e. rotational disks) and software (i.e. file systems), making the application data-, or I/O-intensive. Typically, storing data on clusters, supercomputers, and clouds is achieved by means of distributed or parallel file systems. Such software-infrastructures aggregate multiple disks into a unified (software) storage device accessible via POSIX file system calls from the compute nodes. In this setup, the data is usually transferred from the physical storage device, through the network, transparently to the application. As a consequence, when an application is reading/writing data from/to a distributed or parallel file system, the performance is determined by either disk or network speed.

Historically, reading data over a network was considered slower than reading data from the local disk. Consequently, the *de facto* optimization in data processing was preserving *data locality*. In this way, "computation is sent to the data" in the form of program binaries that are sent to and executed by the machine that stores the data. However, with the advances in networking technology, in the present day, even commodity equipment, such as the 1Gbps Ethernet, guarantees sufficient bandwidth

to render disk locality irrelevant [7]. Therefore, in modern computing platforms that run data processing applications disks are the main bottleneck.

1.1 Many-Task Computing

Together with the increase in data-intensive applications, a new computing paradigm has emerged: many-task computing (MTC) [85]. Scientists and researchers express their complex data analyses as sets of (loosely-coupled) scripts, bags-of-tasks [78], scientific workflows [105], MapReduce [24], or a combination of these. MTC lies at the intersection of high performance computing (HPC) and high-throughput computing (HTC) [12], as typical applications are composed of loosely-coupled tasks that communicate via file system operations, as opposed to the message-passing mechanisms [38] of high-performance computing.

When communication takes place through file system operations application tasks exhibit file dependencies: the output files of certain tasks are input files for other tasks. Therefore, when running MTC applications, there are three distinct types of data: input, communication, and output files. Structurally, MTC applications are composed of large parallel stages, linked by data aggregation and partitioning tasks. The former read data generated by many other tasks, while the latter produce data to be read by many other tasks. Such communication schemes often lead to storage and network traffic imbalance and high variability of the achieved parallelism, and thus to limited scalability.

Due to intrinsic application structure, in MTC, the I/O subsystem accomplishes a dual role. It acts as both storage for the application input and output, but also as a communication layer. As a consequence, not only large amounts of data are analyzed by the application, but also large amounts of data are generated for communication. This puts additional pressure on traditional, disk-based storage systems when running MTC applications, thus limiting performance and scalability.

In this thesis we investigate how distributed or parallel file systems could be improved for achieving a more efficient execution of MTC applications. We believe that the key for optimizing the execution of such applications on large-scale computing infrastructures is to facilitate a seamless and fast file-based communication. Therefore, the storage system needs to be redesigned and tailored according to the MTC application characteristics, while also taking into account the underlying computing infrastructure characteristics. The main research question addressed in this dissertation is the following:

- *How do we tailor-make a storage system in order to optimize the execution of MTC applications?*

1.2 MTC Application Storage Requirements

After analyzing the typical MTC application structure, characteristics and its interaction with the storage system and the underlying computing infrastructure, we have defined a set of *requirements* that a distributed or parallel file system should adhere to when running such applications:

- **R1.** Communication data should benefit from fast I/O to improve application performance. Since communication data is typically temporary data, we can relax the assumption that such files should reside in persistent storage. The only parts of data that need to be persistent are application input and output.

- **R2.** Accessing data should benefit from uniform performance irrespective of data placement. Flattening the storage space simplifies scheduling, and renders locality optimizations irrelevant. Moving the performance optimization aspects from the scheduler to the data storage layer simplifies scheduler logic and improves the load balance of computation, data storage and network links.

- **R3.** The storage system should be flexible and adaptable to different computing infrastructures: clusters, supercomputers, clouds. As opposed to clusters and supercomputers, clouds do not offer performance isolation. As a consequence, the storage system needs to take advantage of platform heterogeneity in order to improve application efficiency and performance.

- **R4.** The storage system should make judicious use of the available resources in the computing infrastructure. Many studies already point out that most computing infrastructures are highly under-utilized. Thus, the storage systems need not add to the under-utilization, but rather improve it and take advantage of poorly-utilized resources, if possible.

1.3 In-Memory Computing

To speed up MTC applications, we believe it is necessary to bypass disk storage altogether when considering temporary communication data. This intermediary data is highly volatile, as the *lifespan* of communication files is typically short: a file only needs to be stored as long as there are still unexecuted tasks that will read it as input. The performance penalty for storing temporary data to disk outweighs the benefit of data persistence (excluding the scenario when disks fail, for which fault-tolerance mechanisms are applicable).

We therefore propose storing application-generated data in main memory. When a set of compute nodes runs a MTC application, the input data will be read from a traditional, disk-based, persistent storage. Then, the communication files will be stored in the compute nodes' main memory aggregated into an in-memory distributed file

Table 1.1: DAS-2 - DAS-5 node comparison.

Cluster Generation	CPU cores	Memory Size (GB)	Disk Size (GB)	Network
DAS-2 (2002)	2 @1Ghz	1	20	Myrinet-2000
DAS-3 (2006)	4 @2.4Ghz	4	250	Myrinet-10G
DAS-4 (2010)	8 @2.4Ghz	24	2000	QDR InfiniBand
DAS-5 (2015)	16 @2.4Ghz	64	8000	FDR InfiniBand

system. When the application finishes executing, the output will be stored into persistent storage.

To assess the feasibility of an in-memory distributed file system for temporary MTC application data, we need to answer the following questions:

- (1) How much faster is memory compared to disks?
- (2) Is there enough memory in modern large-scale computing infrastructures?
- (3) Is the network the new performance bottleneck? How does network bandwidth compare to memory bandwidth?

To answer these questions we analyze the Distributed ASCI Supercomputer (DAS) [10] evolution over time. For simplicity, we will only refer to the nodes' characteristics of the DAS cluster located at the Vrije Universiteit Amsterdam[1].

Table 1.1 shows the node equipment for the DAS generations 2 to 5. Over a 13-year span, the number of cores in the compute nodes has increased eight times. However, the memory size exhibits a 64-fold increase, while the disk sizes have increased with two orders of magnitude, from 20 GB to 8 TB. The rate at which the theoretical network bandwidth has increased is similar to the increase in memory sizes.

Even though the memory sizes are two orders of magnitude smaller than the disk sizes, considering the size of the DAS-4 (74 nodes) and DAS-5 (68 nodes) clusters, the total memory of the clusters - 1.7 TB and 4.3 TB - is sufficient for a large class of applications. Moreover, considering the rate at which memory sizes increase, in the future we expect to be able to support applications that store orders of magnitude more data.

Table 1.2 shows the memory, disk and network bandwidth achieved by the nodes of the DAS clusters. The numbers were obtained by running the *stream* [103] benchmark for memory and the *iozone* [76] benchmark for disk. For the network bandwidth, the DAS-2 and DAS-3 measurements were performed using the MPI bandwidth benchmark [68], while for DAS-4 and DAS-5 we used *qperf*.

We notice that the memory bandwidth has increased ten times in 13 years. For disks, we notice a nearly-similar behaviour. However, the network bandwidth for

[1] `http://www.vu.nl`

Table 1.2: DAS-2 - DAS-5 Memory vs. Disk vs. Network Bandwidth.

Cluster Generation	Memory (GB/s)	Disk (GB/s)	Network (GB/s)
DAS-2	5.5	0.040	0.16
DAS-3	9	0.058	0.95
DAS-4	22	0.187	6.2
DAS-5	56	0.350	12

DAS-5 is two orders of magnitude higher than for DAS-2, increasing from 160MB/s to 12GB/s. Using the values from Table 1.2, we plotted Figure 1.1 that shows the ratio between memory bandwidth versus disk and network bandwidth for all DAS generations. It is interesting to notice that the ratio of memory-to-disk bandwith has remained constant over the years, with a two orders of magnitude advantage of memory over disk. However, when comparing the ratio of memory-to-network bandwith, the results are encouraging. If for DAS-2 the difference in memory and network bandwidth was approximately two orders of magnitude, for DAS-4 and DAS-5 the ratio of memory-to-network bandwidth is lower than an order of magnitude. Considering these results, we believe that in the future the gap will become even smaller and we will be able to access remote memory at the same bandwidth with local memory.

In conclusion, the gap between memory and disk I/O highly encourages storing MTC application communication data in the former. The downside of storing data in the main-memory of the compute nodes is that disks still offer orders of magnitude higher capacities. Therefore, at the moment only applications whose data fit into memory can benefit from in-memory distributed file systems. An alternative could be to split a larger application into several smaller applications that analyze partial views of the data. These smaller applications would then be run in a sequence, ensuring that their data fits in memory, and benefits from its superior I/O performance. Even though currently, for in-memory distributed file systems, the network could be considered the bottleneck, as it is slower than memory, we consider networks sufficiently fast for current MTC applications.

1.4 Related Approaches

We identified two approaches similar to our proposal of storing MTC application data in in-memory distributed file systems: *burst buffers*, and *in-situ processing*. Both try do bridge the gap between the computing capacities and the traditional, disk-based, I/O systems of modern, large-scale computing infrastructures. Burst buffers [66, 93] provide a fast cache between application that exhibit bursty I/O patterns and the parallel file system. This fast cache absorbs the I/O of the application and then transfers, asynchronously, the data to persistent, disk-based storage.

In-situ processing techniques and middleware, presented in [129, 128, 1, 27] re-

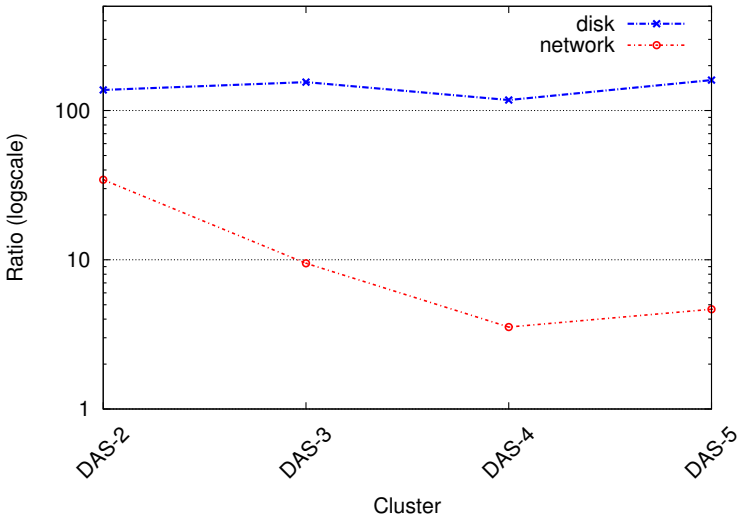

Figure 1.1: Ratio between memory bandwidth vs. disk and network bandwidth.

fer to analyzing or visualizing simulation-generated data just as it had been created, before being written to the parallel file system. Post-processing the data from disk storage would incur large performance penalties. Therefore, in-situ computing middleware processes data while it still resides in memory and is thus accessible much faster.

Even though in-memory distributed file systems for MTC applications are related in purpose to burst buffers and in-situ processing middleware, what differentiates our approach is the MTC application structure and characteristics. MTC applications are not bursty, and they don't need their communication data stored in persistent storage. Also, as opposed to burst buffers (that run generally on dedicated nodes), our approach stores data in the memory of the compute nodes running the application. Moreover, MTC applications communicate via (POSIX) file system calls. In-situ processing middleware offers special libraries and APIs for retrieving data from memory, bypassing file system calls. In conclusion, our approach is complementary to both techniques and it is applicable to another class of applications.

1.5 Thesis Contributions and Outline

In this dissertation, we propose an incremental design of an in-memory distributed file system for MTC application that addresses the four requirements (**R1-R4**) introduced previously. In Chapter 2, we introduce an in-memory distributed file system, MemFS, that adheres to **R1** and **R2**. Building on MemFS, in Chapter 3 we address **R4** from an application perspective, improving resource utilization for in-

memory distributed file systems when running MTC applications. In Chapter 4 we address **R3**, preparing our storage system to be deployed on platforms with variable network performance. In Chapter 5, we address **R4** from the computing infrastructure perspective. We create a system that takes advantage of under-utilized cluster reservations to improve resource utilization. Finally, in Chapter 6 we conclude the dissertation and identify possible future work directions.

1.5.1 Chapter 2: MemFS - The No-Locality Approach

To alleviate the performance bottleneck induced by disk-based storage solutions, state-of-the-art proposes using locality-based in-memory distributed file systems. However, we notice that the locality-based approach has two disadvantages: it imbalances the storage of the system nodes, and the network traffic. This limits the applicability and performance of in-memory runtime distributed file systems for MTC applications.

To overcome these bottlenecks, we design **MemFS**, a locality-agnostic in-memory storage system. MemFS distributes data evenly across compute nodes taking advantage of high-speed modern interconnects such as InfiniBand or 10G Ethernet. Our system outperforms the locality-based approach and is applicable in both clusters and clouds with premium networks. This work has been published in:

> **MemFS: an In-Memory Runtime File System with Symmetrical Data Distribution**, Alexandru Uta, Andreea Sandu, and Thilo Kielmann. In *IEEE International Conference on Cluster Computing (CLUSTER), pages 272–273, IEEE, 2014. (poster paper, best poster award).*
>
> **Overcoming Data Locality: An In-memory Runtime File System with Symmetrical Data Distribution**, Alexandru Uta, Andreea Sandu, and Thilo Kielmann. In *Future Generation Computer Systems, 54:144–158, 2016.*

1.5.2 Chapter 3: MemEFS - Elasticity Improves Resource Utilization

One limitation of current in-memory distributed file systems lies in the fact that they are typically deployed onto a static number of compute resources. Also, the application data footprint exhibits large degrees of variability during runtime. Hence, the user has to over-provision resources, in advance, to accommodate the peak storage demand of the application. This results in largely under-utilizing the acquired resources.

To improve resource utilization, we designed MemEFS, an elastic counterpart of our previous work. MemEFS is able to improve resource utilization efficiency by dynamically adding or removing compute resources based on the application storage demand. Furthermore, MemEFS has been selected as an IEEE TCSC Scale

Challenge finalist, showing that it can achieve **three dimensions of scalability** - horizontal, vertical and elastic. This work has been published in:

> **Scalable In-Memory Computing**, Alexandru Uta, Andreea Sandu, Stefania Costache, and Thilo Kielmann. In *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on, pages 805–810. IEEE, 2015. (IEEE TCSC Scale Challenge Finalist)*

> **MemEFS: an Elastic In-Memory Runtime File System for eScience Applications**, Alexandru Uta, Andreea Sandu, Stefania Costache, and Thilo Kielmann. In *e-Science (e-Science), 2015 IEEE 11th International Conference on, pages 465–474. IEEE, 2015. (best eScience service or project award)*

1.5.3 Chapter 4: MemEFS - Network-Awareness

To tackle the *data deluge* induced by the ever-increasing amounts of data generated by scientific domains (e.g. bioinformatics, climate modeling, astronomy), cost-effective compute platforms need to be deployed. This could be achieved by extending private clusters with public cloud computing resources. However, many studies point out that the networks of public clouds are plagued by large degrees of variability due to collocation and virtualization overheads.

To address this issue, we have designed a network-adaptation mechanism for MemEFS. This mechanism monitors the bandwidth capacities of the worker nodes, and balances the data distribution accordingly. Compared to its network-agnostic counterpart, the network-adapted MemEFS largely improves the runtime of MTC applications on clouds. This work is currently under submission in:

> **MemEFS: A Network-Aware In-Memory Distributed File System**, Alexandru Uta, Ove Danner, Ana-Maria Oprescu, Andreea Sandu, Stefania Costache, and Thilo Kielmann. *under submission in: Future Generation Computer Systems*

1.5.4 Chapter 5: MemFSS - Memory Scavenging

Current private compute clusters suffer from large degrees of under-utilization. Reports show that up to 50% of the memory is unused, while there is also enough available network bandwidth. To take advantage of these unused resources and increase total cluster utilization efficiency, we propose a **memory scavenging** framework - MemFSS. Using this mechanism, MemFSS is able to seamlessly scavenge for available memory in other users' node reservations. Therefore, the total cluster application throughput and resource utilization are higher, and users of MemFSS do not need to over-provision their cluster reservations. Furthermore, the performance

penalty incurred by tenant applications due to MemFSS is negligible. This work has been published in:

> **Towards Resource Disaggregation - Memory Scavenging for Scientific Workloads**, Alexandru Uta, Ana-Maria Oprescu, and Thilo Kielmann. In *IEEE International Conference on Cluster Computing (CLUSTER), pages 100-109, IEEE, 2016.*